12/4/2015 EE-334 S.6.5.1

Different Groups of Instructions in Instruction Set of 8088/8086:

Instruction Set:-

- The instruction set of a microprocessor defines the basic operations that a programmer can make the device perform.
- The instruction set of 8088/8086 microprocessor has 117 basic instructions.

AAA	CMPSB	JAE	JNBE	JPO	MOV	RCR	SCASB
AAD	CMPSW	JB	JNC	JS	MOVSB	REP	SCASW
AAM	CWD	JBE	JNE	JZ	MOVSW	REPE	SHL
AAS	DAA	JC	JNG	LAHF	MUL	REPNE	SHR
ADC	DAS	JCXZ	JNGE	LDS	NEG	REPNZ	STC
ADD	DEC	JE	JNL	LEA	NOP	REPZ	STD
AND	DIV	JG	JNLE	LES	NOT	RET	STI
CALL	HLT	JGE	JNO	LODSB	OR	RETF	STOSB
CBW	IDIV	JL	JNP	LODSW	OUT	ROL	STOSW
CLC	IMUL	JLE	JNS	LOOP	POP	ROR	SUB
CLD	IN	JMP	JNZ	LOOPE	POPA	SAHF	TEST
CLI	INC	JNA	JO	LOOPNE	POPF	SAL	XCHG
CMC	INT	JNAE	JP	LOOPNZ	PUSH	SAR	XLATB
CMP	INTO	JNB	JPE	LOOPZ	PUSHA	SBB	XOR
	IRET				PUSHF		
	JA				RCL		

- The wide range of operands and addressing modes permitted for use with these instructions executable at the machine code level.
- The instruction set can be divided into a number of groups of functionally related instructions.

1. Data Transfer Instructions.
2. Isolated I/O Instructions.
3. Arithmetic Instructions.
4. Logic Instructions.
5. Compare Instructions.
6. Shift Instructions.
7. Rotate Instructions.
8. Flag control Instructions.
9. Jump Instructions.
10. Subroutine handling Instructions.
11. Stack operations Instructions.
12. Loop handling Instructions.
13. String handling Instructions.

DATA TRANSFER INSTRUCTIONS

GENERAL – PURPOSE BYTE OR WORD TRANSFER INSTRUCTIONS	SIMPLE INPUT AND OUTPUT PORT TRANSFER INSTRUCTION	SPECIAL ADDRESS TRANSFER INSTRUCTION	FLAG TRANSFER INSTRUCTIONS
MOV PUSH POP XCHG XLAT/XLATB	IN OUT	LEA LDS LES	LAHF SAHF PUSHF POPF

ARITHMETIC INSTRUCTIONS

ADDITION INSTRUCTIONS	SUBTRACTION INSTRUCTIONS	MULTIPLICATION INSTRUCTIONS	DIVISION INSTRUCTIONS
ADD ADC INC AAA DAA	SUB SBB DEC NEG CMP AAS DAS	MUL IMUL AAM	DIV IDIV AAD CBW CWD

BIT MANIPULATION INSTRUCTIONS

LOGICAL INSTRUCTIONS	SHIFT INSTRUCTIONS	ROTATE INSTRUCTIONS
NOT AND OR XOR TEST	SHL / SAL SHR SAR	ROL ROR RCL RCR

STRING INSTRUCTIONS

STRING INSTRUCTIONS
REP
REPE / REPZ
REPNE / REPNZ
MOVS / MOVSB / MOVSW
COMPS / COMPSB / COMPSW
SCAS / SCASB / SCASW
LODS / LODSB / LODSW
STOS / STOSB / STOSW

PROGRAM EXECUTION TRANSFER INSTRUCTIONS

UNCONDITIONAL TRANSFER INSTRUCTIONS	CONDITIONAL TRANSFER INSTRUCTIONS		ITERATION CONTROL INSTRUCTIONS	INTERRUPT INSTRUCTIONS
CALL	JA / JNBE	JLE / JNG	LOOP	INT
RET	JAE / JNB	JNC	LOOPE / LOOPZ	INTO
JMP	JB / JNAE	JNE / JNZ	LOOPNE / LOOPNZ	IRET
	JBE / JNA	JNO	JCXZ	
	JC	JNP / JPO		
	JE / JZ	JNS		
	JG / JNLE	JO		
	JGE / JNL			
	JL / JNGE	JP / JPE		
		JS		

PROCESS CONTROL INSTRUCTIONS

FLAG SET / CLEAR INSTRUCTIONS	EXTERNAL HARDWARE SYNCHRONIZATION INSTRUCTIONS
STC	HLT
CLC	WAIT
CMC	ESC
STD	LOCK
CLD	NOP
STI	
CLI	

Note:

Here, I explain them under:

2-operand instructions — Ex. ADD **BX**, **CX**

1-operand instructions — Ex. PUSH **SI**

0-operand instructions: — Ex. DAA

Data Transfer Instructions: -

- These instructions are used to transfer data from source to destination.
- The operand can be a constant, memory location, register or I/O port address.

* Move byte or word (MOV) Instruction.
* Exchange byte or word (XCHG) Instruction.
* Translate byte (XLAT) Instruction.
* Load effective address (LEA) Instruction.
* Load data segment (LDS) Instruction.
* Load extra segment (LES) Instruction.

Note: We are going to investigate these instructions in details

Mnemonic	Function
MOV	Move byte or word to register or memory
IN, OUT	Input byte or word from port, output word to port
LEA	Load effective address
LDS, LES	Load pointer using data segment, extra segment
PUSH, POP	Push word onto stack, pop word off stack
XCHG	Exchange byte or word
XLAT	Translate byte using look-up table

MOV Instruction

- MOV instruction is used to transfer a byte or a word of data from a source operand to a destination operand.
- It does not modify flags.

Mnemonic	Meaning	Format	Operation	Flags affected
Mov	Move	Mov D,S	(s) ⟶ (D)	None

Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg reg	Reg 16
Seg reg	Mem 16
Reg 16	Seg reg
Memo	Seg reg

NO MOV

Mem ⟶ Mem
Imm ⟶ Seg Reg
Seg Reg ⟶ Seg reg

```
MOV     reg8, r/m/i8
MOV     mem8, reg8
MOV     mem8, BYTE imm8

MOV     reg16, r/m/i16
MOV     mem16, reg16
MOV     mem16, WORD imm16

MOV     r/m16, segreg
MOV     segreg, r/m16
```

MOV d,s

d=destination (register or effective memory address),

s=source (immediate data, register or memory address)

- MOV can transfer data from:
  - any register to any register (except CS register)
  - memory to any register (except CS)
  - immediate operand to any register (except CS)
  - any register to a memory location
  - immediate operand to memory
- MOV cannot perform memory to memory transfers (must use a register as an intermediate storage).
- MOV moves a word or byte depending on the operand bit-lengths; the source and destination operands must have the same bit length.
- MOV cannot be used to transfer data directly into the CS register.

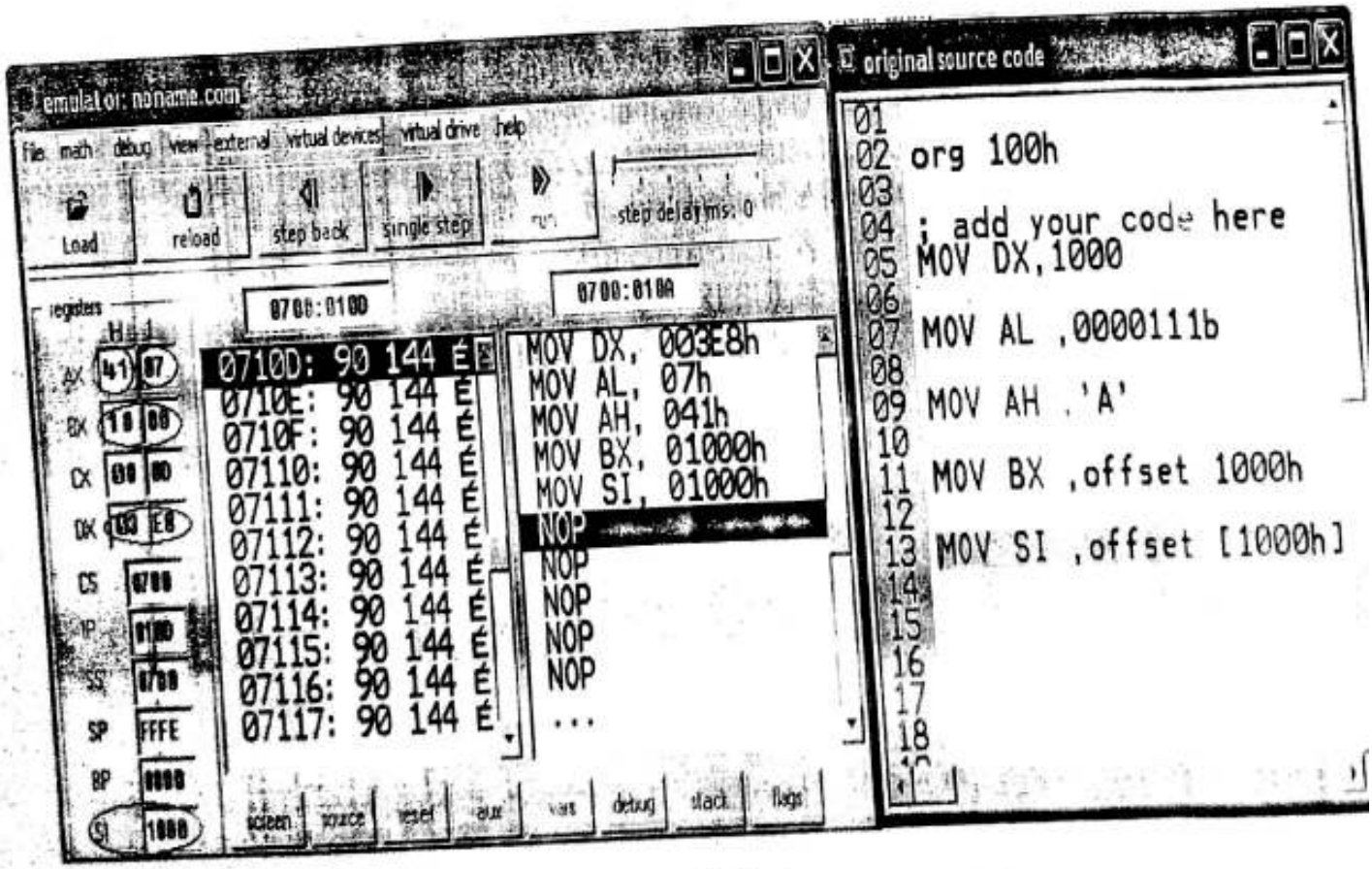
- The number of clock cycles needed for an instruction depends on the addressing mode.
- For 8088/8086 systems, calculation of the *effective address* (16 bits) of the operand can take several clock cycles (these extra clock cycles are not needed for 80286 or later microprocessors):

Examples Extra clock cycles needed for calculating effective address:

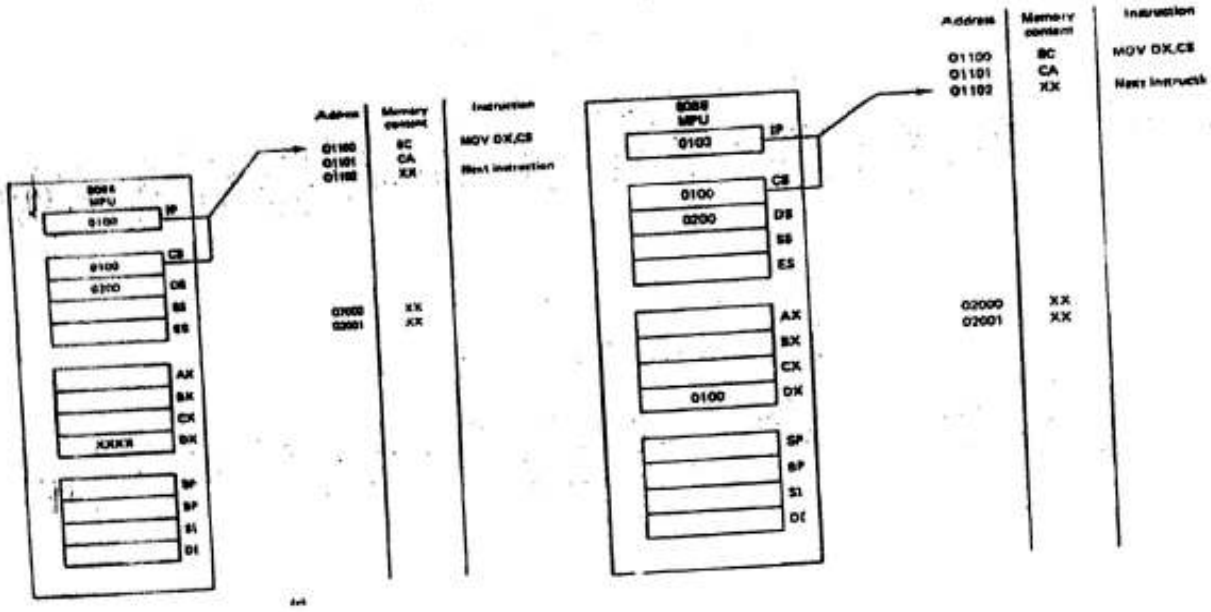
Addressing mode	Example	Clock
register indirect	MOV CL,[DI]	5
Direct	MOV CL,FRED	3
Based index	MOV BL,[BP+DI]	7
Based index (displacement)	MOV CX,[BP+DI+FRED]	11

EX

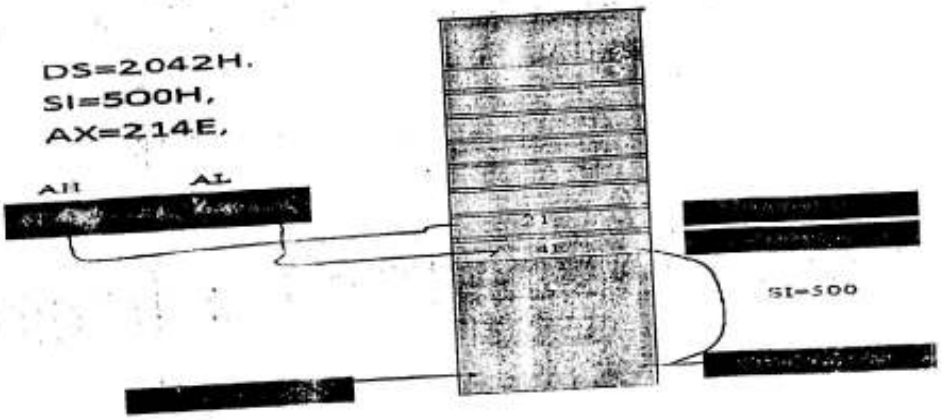
➢ MOV CX, 037AH	Put immediate number 037AH to CX
➢ MOV BL, [437AH]	Copy byte in DS at offset 437AH to BL
➢ MOV AX, BX	Copy content of register BX to AX
➢ MOV DL, [BX]	Copy byte from memory at [BX] to DL
➢ MOV DS, BX	Copy word from BX to DS register
➢ MOV RESULT [BP], AX	Copy AX to two memory locations; AI to the first location, AH to the second; EA of the first memory location is sum of the displacement represented by RESULTS and content of BP. Physical address = EA + SS.
➢ MOV ES: RESULTS [BP], AX	Same as the above instruction, but physical address = EA + ES, because of the segment override prefix ES



MOV DX,CS



MOV [SI],AX



EXAMPLE

What is the effect of executing the instruction

MOV CX, [SOURCE_MEM]

Where SOURCE_MEM equal to 20_{16} is a memory location offset relative to the current data segment starting at $1A000_{16}$.

Solution:

$((DS)0+20_{16}) \rightarrow (CL)$

$((DS)0+20_{16}+1_{16}) \rightarrow (CH)$

Therefore CL is loaded with the contents held at memory address

$1A000_{16} + 20_{16} = 1A020_{16}$

and CH is loaded with the contents of memory address

$1A000_{16} + 20_{16} + 1_{16} = 1A021_{16}$

Notes:-

In order to say the compiler about data type,these prefixes should be used:
BYTE PTR - for byte.
WORD PTR - for word (two bytes).

For example

MOV AL, byte PTR [SI]

MOV AX, WORD PTR [SI]

Emu8086 supports shorter prefixes as well:
b. - for BYTE PTR
w. - for WORD PTR

```
MOV Al ,b.[SI]
MOV AX ,W.[SI]
```

EX

Use the move instruction to copy the contents of memory offset 0300H to memory offset 0500H.

MOV AL,[0300H]
MOV [0500H],AL

EX

Use the move instruction to copy the contents of memory offset 0300H and 0301H to memory offset 0500H and 0501H.

MOV AX , [0300H]
MOV [0500H] , AX

EX

Write a program to store the data 15H , 75H into memory locations 0055H,01FCH. Assume DS=7000H.

```
MOV AX , 7000H
MOV DS , AX
MOV BX , 0055H
MOV [BX] , 15H
MOV BX , 01FCH
MOV [BX] , 75H
```

EX

Write a program to **exchange** contents of memory location offset 3F00H with that of memory location offset FFC1H.

```
MOV AL,[3F00H]
MOV BL,[FFC1H]
MOV [3F00H],BL
MOV [FFC1H],AL
```

XCHG Exchange Instruction

- Can exchange AX with a 16 bit register.
- Can exchange an 8 bit register with another 8 bit register or an 8 bit byte in memory.
- Can exchange a 16 bit register with another 16 bit register or a 16 bit word in memory.
- It cannot exchange two memory locations directly.

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHG S,D	(S) ↔ (D)	None

Destination	Source
Accumulator	Reg 16
Memory	Register
Register	Register
Register	Memory

Example: XCHG [1234H], BX

[illegible]

[illegible]
[illegible]

XCHG AX,BX AX ↔ BX

XCHG AL,BH AL ⇄ BH

XCHG [SI],DX [SI] ↔ DX

EX:

For the data shown, what is the result of executing XCHG [SUM], BX ?

SUM=1234H, DS=1200

Solution:

Execution of this instruction performs the operation

((DS)0 + SUM) ↔ (BX)

PA = 12000H + 1234H = 13234H

Then,

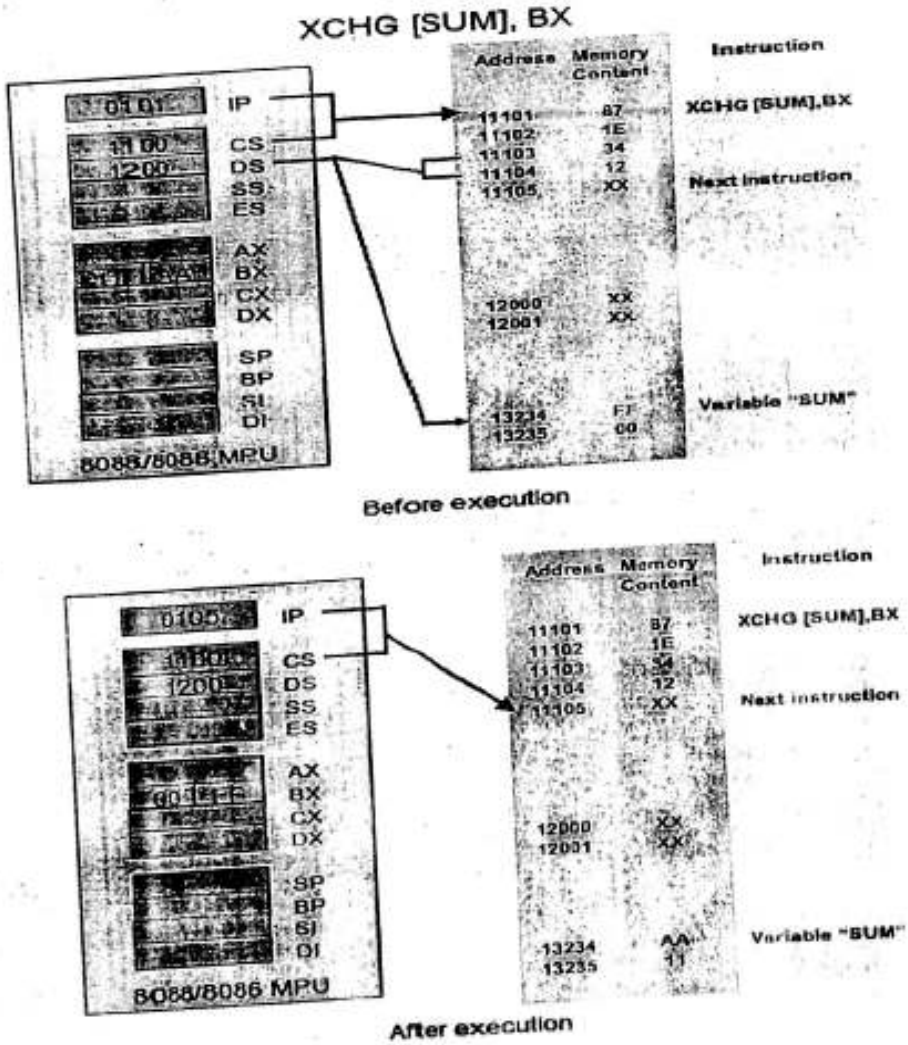
[13234H]↔ (BL)

[13235H]↔ (BH)

Result: (BX) = 00FFH ,

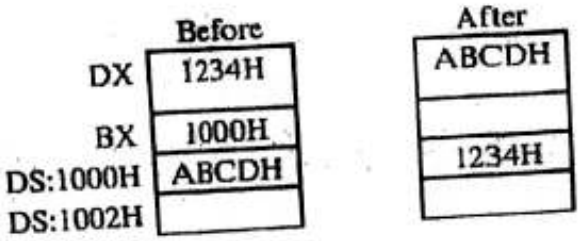
[13234] = AA

[13235] = 11



Exchange Instruction

XCHG DX, [BX]



Ex .

```
MOV AL, 5
MOV AH, 2
XCHG AL, AH    ; AL = 2, AH = 5
XCHG AL, AH    ; AL = 5, AH = 2
```

C	Z	S	O	P	A
unchanged					

EX

```
MOV BX, 01FCH
MOV [BX],50H
MOV AL ,90H
XCHG AL,[BX]
```

EX

```
MOV AX,5000H
MOV DS,AX
MOV  BX,0100H
MOV [BX],0F1CH
MOV CX ,5030H
XCHG CX,[BX]
```

LEA, LDS, LES Instructions

The LEA, LDS, LES instructions provide the ability to manipulate memory addresses by loading either a 16-bit offset address into a general purpose register or a register together with a segment address into either ES or DS.

LEA	Load Effective Address	LEA Reg16,EA	Reg16 ← EA	None
LDS	Load Register And DS	LDS Reg16,MEM32	Reg16 ← (Mem32) DS ← (Mem32+2)	None
LES	Load Register and ES	LES Reg16,MEM32	Reg16 ← (Mem32) ES ← (Mem32+2)	None

LEA Destination, Source

- Transfers the offset address of source (must be a memory location)to the destination register.

(REG = address of memory offset)

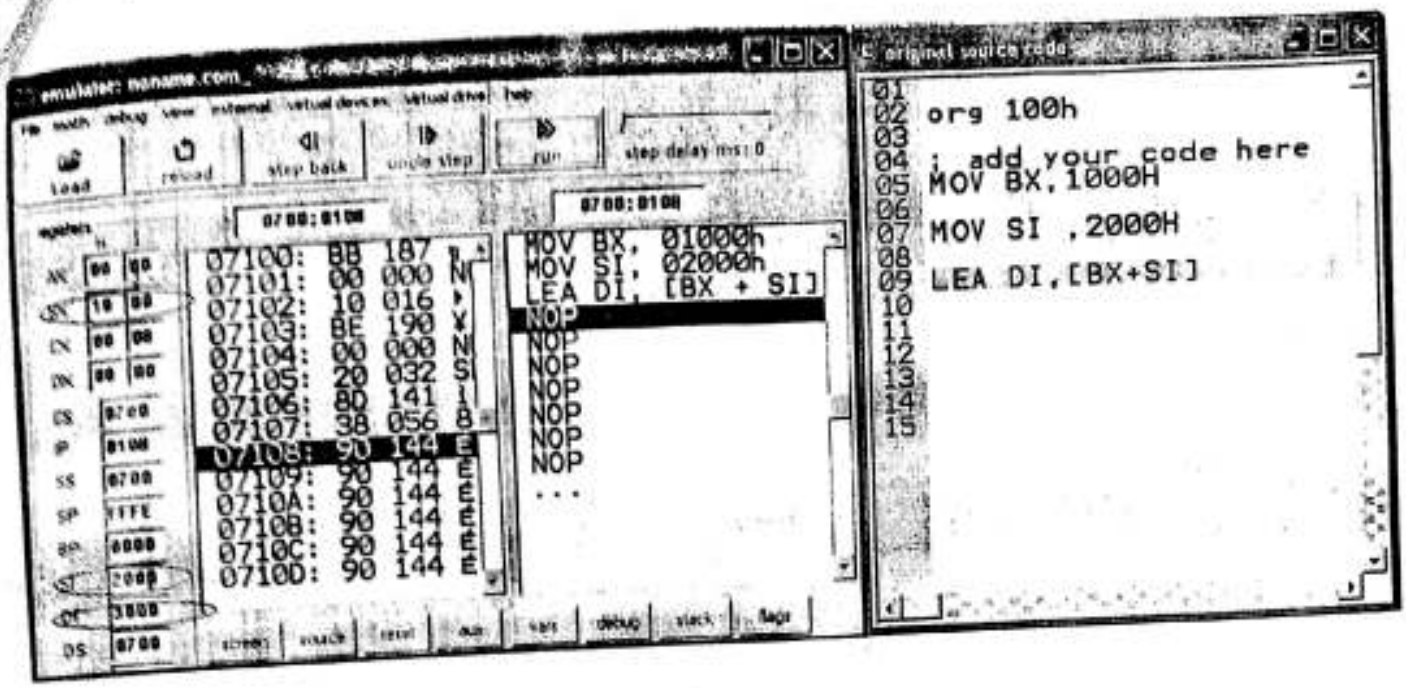
- It loads a 16-bit register with the offset address of the data specified by the Source

EX:

LEA BX, [DI]

This instruction loads the contents of DI (offset) into the BX register.

- It does not modify flags.




```
LEA     BX,NUMB     ;NUMB is assumed to point at
                    the byte at location 10FD in the
                    data segment.
```

BL ← FD
BH ← 10

Note:

* LEA BX,NUMB = MOV BX,OFFSET NUMB

```
LEA     BX,[DI]     ; BX ←  DI
MOV     BX,[DI]     ; BX ←  [DI]
```

LEA bx, SI
LEA bx, [SI]
LEA bx, offset[SI]
LEA bx, offset SI

By comparing LEA with MOV, it's observed that:-

- **LEA BX , [DI]** load the offset address specified by [DI] (contents of DI) into BX register.

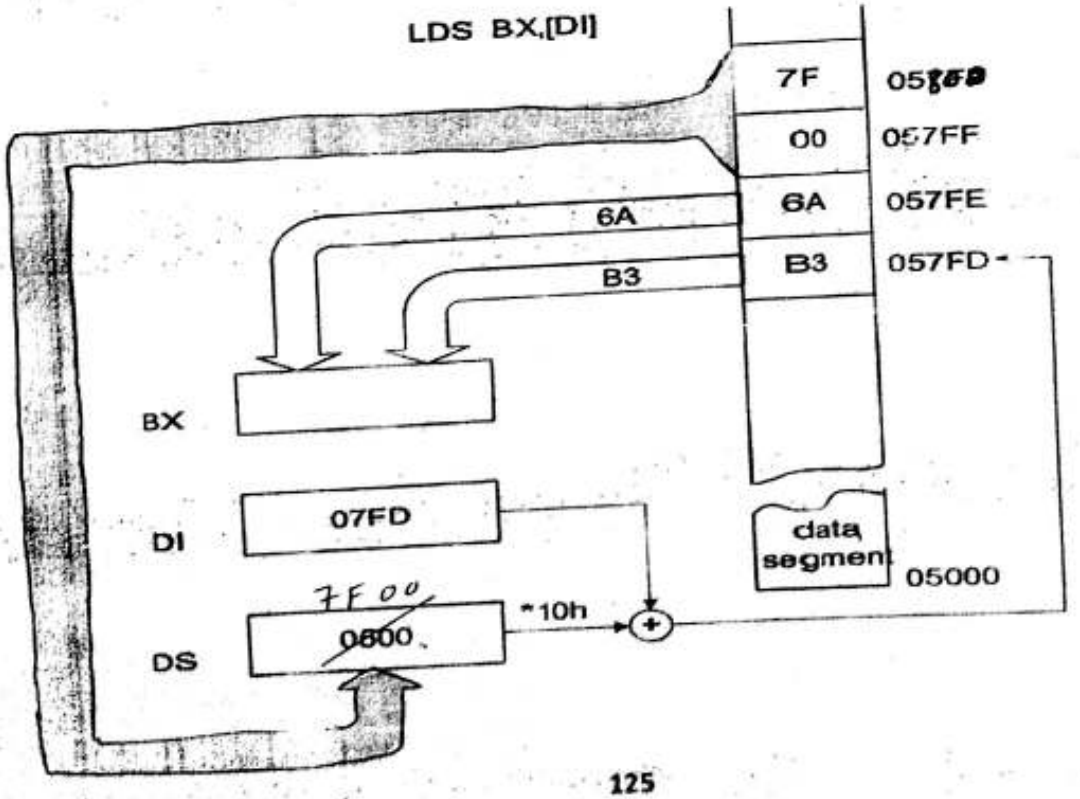
MOV BX , [DI] load the data stored at the memory location addressed by [DI]into register [DI].

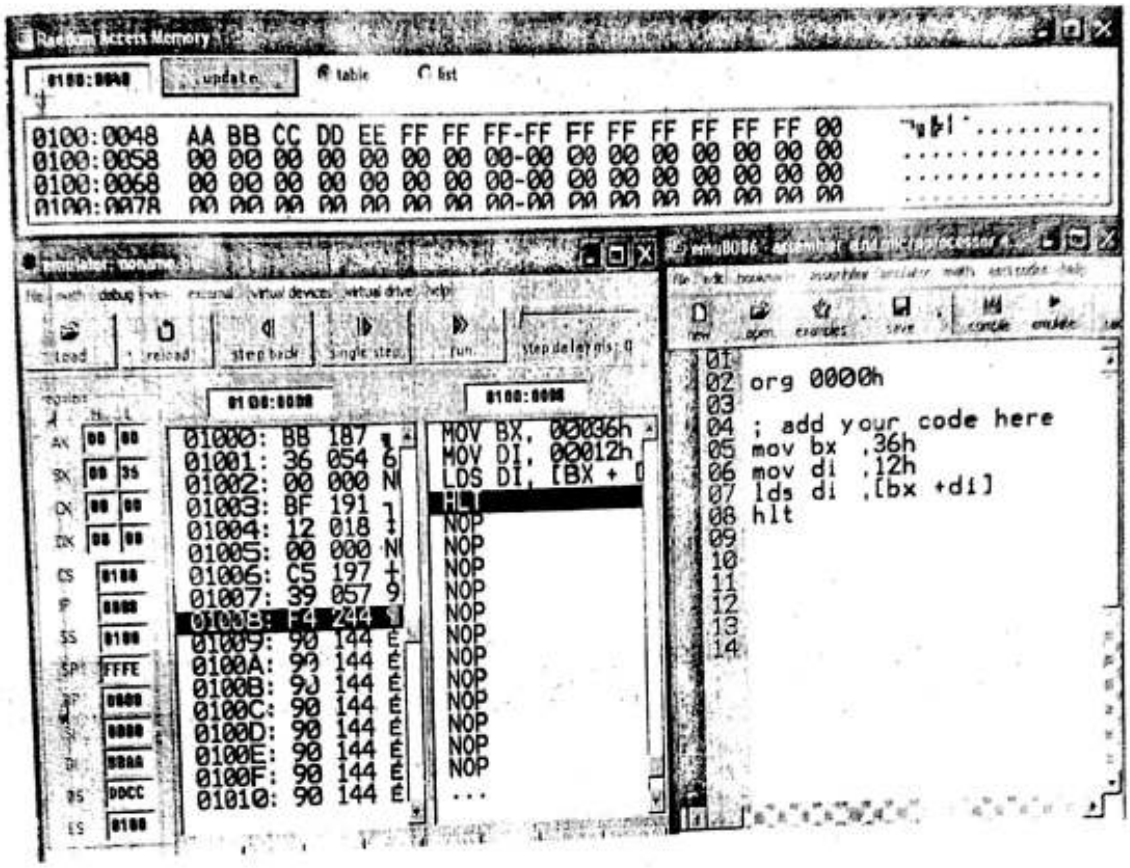
LDS Destination, Source

- Load 4-byte data (pointer) in memory to two 16-bit registers. It loads 32-bit pointer from memory source to destination register and DS.
- It does not modify flags
- The offset is placed in the destination register and the segment is placed in DS.
- Source operand gives the memory location.
- The first two bytes are copied to the register specified in the destination operand; the second two bytes are copied to register DS.

```
REG = First word
DS = Second Word
```

➢ LDS BX, [4326H] — Copy content of memory at displacement 4326H in DS to BL, content of 4327H to BH. Copy content at displacement of 4328H and 4329H in DS to DS register.

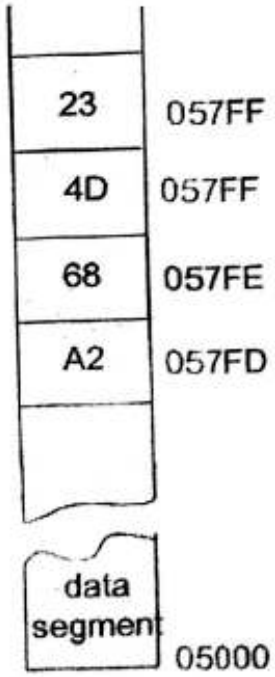




Example:

Assume DS = 500h, SI = 7D0h, BX = 2Dh, determine the contents of all the affected registers if the following instruction is executed:

LDS DI,[BX+SI]



LES Destination, Source

- It is identical to LDS except that the second two bytes are copied to ES
- It does not modify flags.

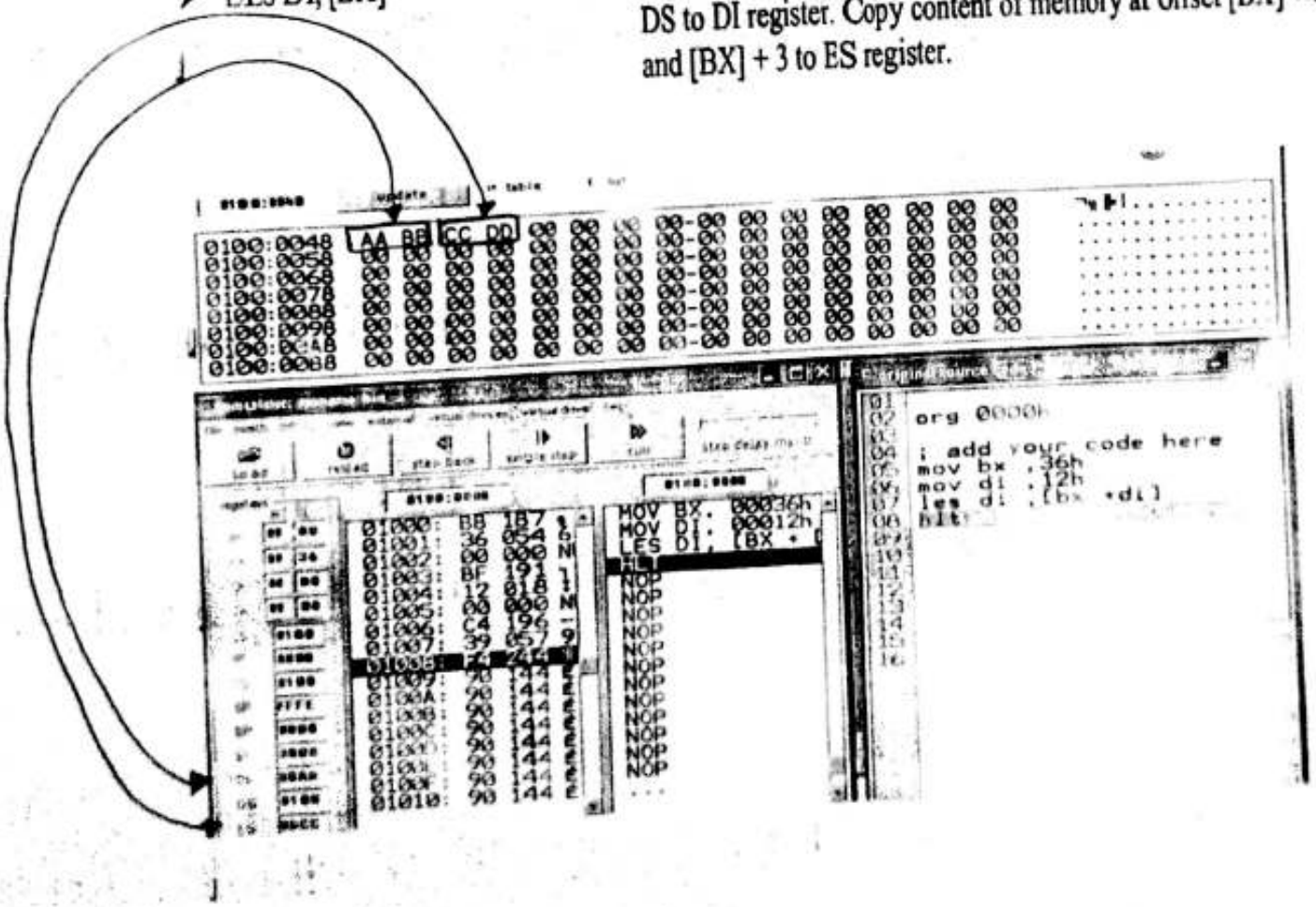
```
REG = first word
 ES = second word
```

➢ LES BX, [789AH]

Copy content of memory at displacement 789AH in DS to BL, content of 789BH to BH, content of memory at displacement 789CH and 789DH in DS is copied to ES register.

➢ LES DI, [BX]

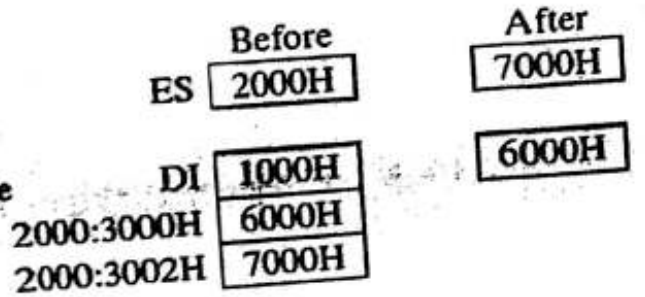
Copy content of memory at offset [BX] and offset [BX] + 1 in DS to DI register. Copy content of memory at offset [BX] + 2 and [BX] + 3 to ES register.



2-Operand instruction to load ES and an Address Register from memory

LES AR, M32

LES DI,[3000H]



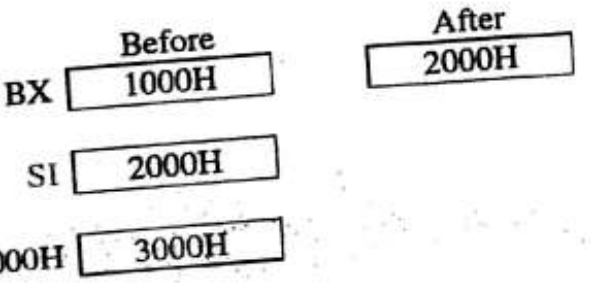
Loads ES and DI using single instruction

2-Operand instruction to load an Effective address into an Address Register

LEA AR, a16

EX

LEA BX, [SI]



LEA BX, [SI] functionally same as

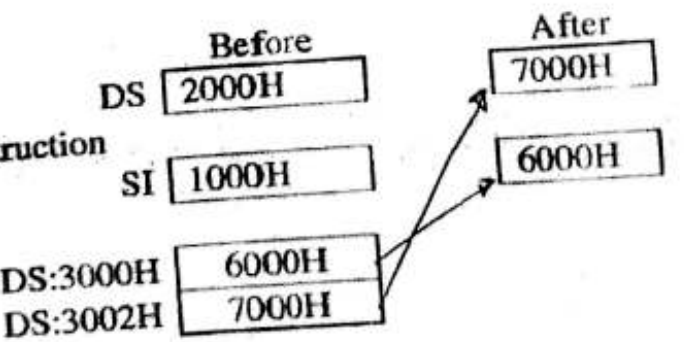
MOV BX,SI

2-Operand instruction to load DS and an Address Register from memory

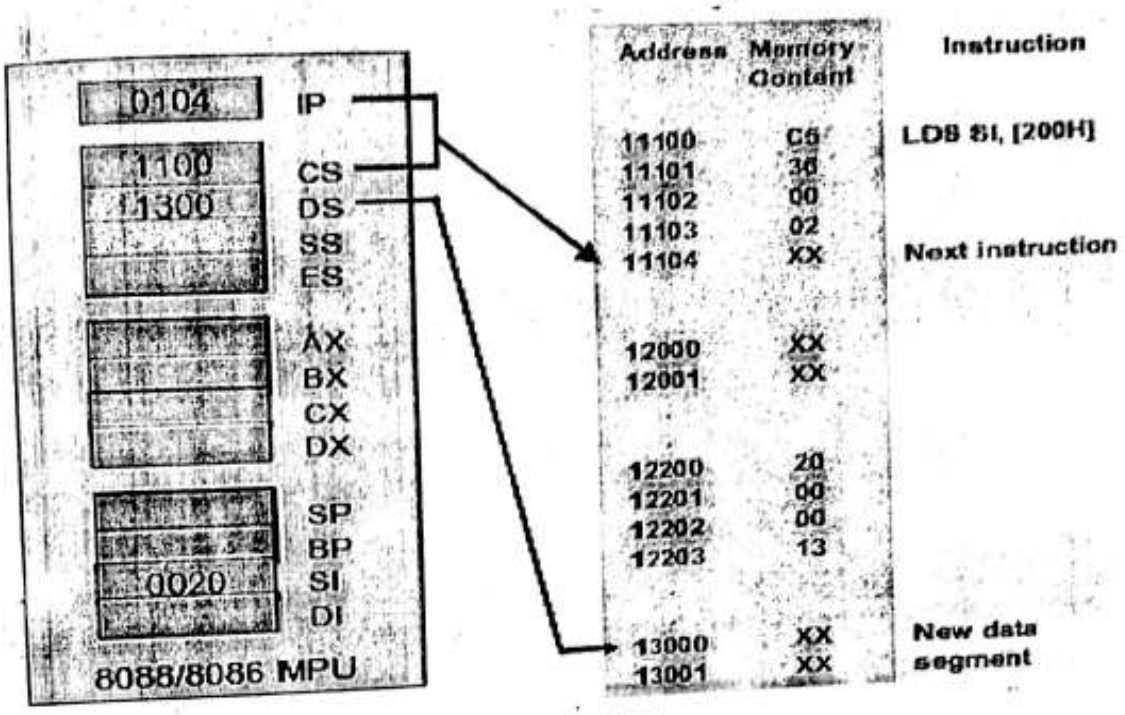
LDS AR, M32

EX

LDS SI,[3000H]

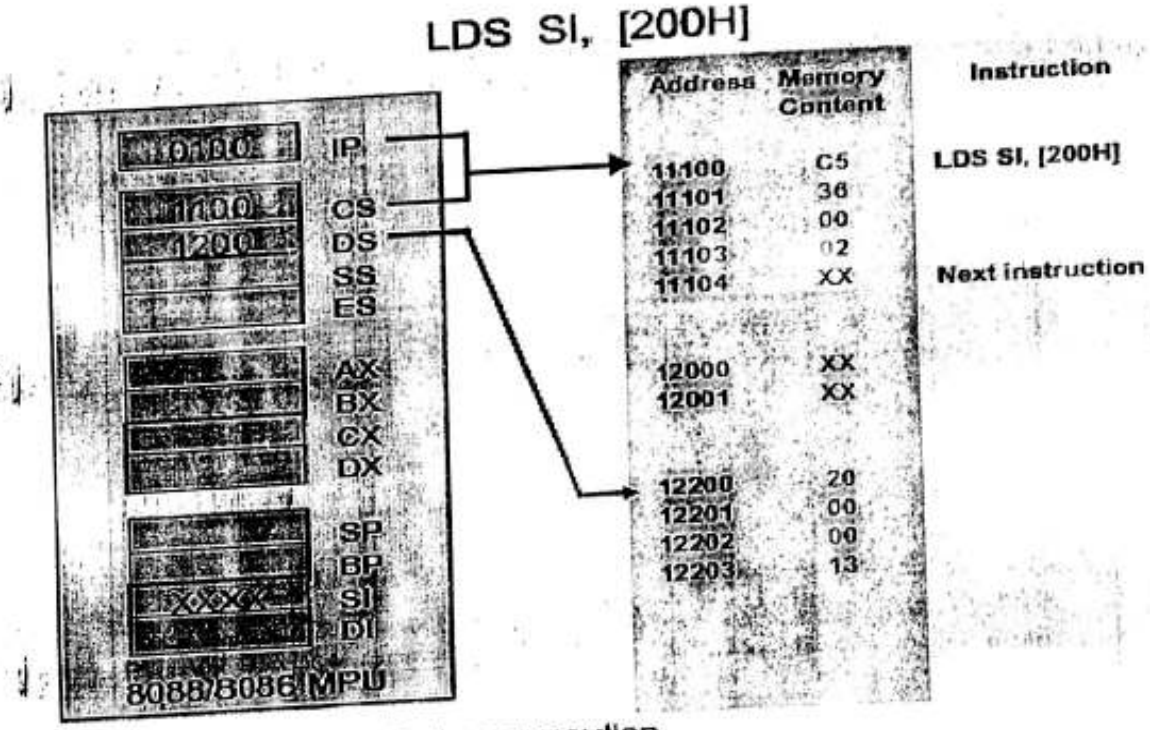


Loads DS and SI using single instruction



After execution

LDS SI, [200H]



Before execution

Example 1 : Assuming that BX=20H ,DI=1000H , DS=1200H and the following memory contents:

Memory	12200	12201	12202	12203	12204
Content	11	AA	BE	FF	22

What result is produced in the destination operand by execution the following instruction?

a- LEA SI,[DI+BX+5] b- LDS SI,[200]

Solution :

a- EA =1000+20+5=1025 then (SI)=1025

b- PA=DS:EA=DS*10+EA=1200*10+200=12200

(SI)=AA11H and (DS)=FFEEH

The LDS AND LES instructions

- LDS and LES load a 16-bit register with offset address retrieved from a memory location.
- then load either DS or ES with a segment address retrieved from memory.

```
LDS     BX,DWORD PTR[SI]
        BL  <---  [SI];
        BH  <---  [SI+1]
        DS  <---  [SI+3;SI+2];     in the data segment

LES     BX,DWORD PTR[SI]
        BL  <---  [SI];
        BH  <---  [SI+1]
        ES  <---  [SI+3;SI+2];     in the extra segment
```

LAHF

It does not modify flags

SAHF

It modifies flags: AF, CF, PF, SF, ZF

The LAHF and SAHF instructions

LAHF	AH	$\leftarrow$ $Flags_{low}$
SAHF	$Flags_{low}$	$\leftarrow$ AH

EX

Write ALP that save the contents of 8086's flags in memory location having an offset 1212 H and then to reload the flags from the contents of the memory location having an offset 2121 H.

```
LAHF                    ; Load AH from flags
MOV [1212], AH          ; Move the contents of AH to memory location
                        ; pointed to by offset 1212 H

MOV AH, [2121]          ; Move the contents of memory locations pointed
                        ; to offset 2121H to AH

SAHF                    ; Store AH into flags
HLT                     ; Stop
```


Page 131 X

IN/OUT Instructions

- An **IN** instruction transfers data from an external I/O device to AL or AX.
- An **OUT** instruction transfers data from AL or AX to an external I/O device.
- The I/O device address is called port address.
- In direct addressing mode the address is a single byte.
- In indirect addressing mode the address is two bytes in DX.

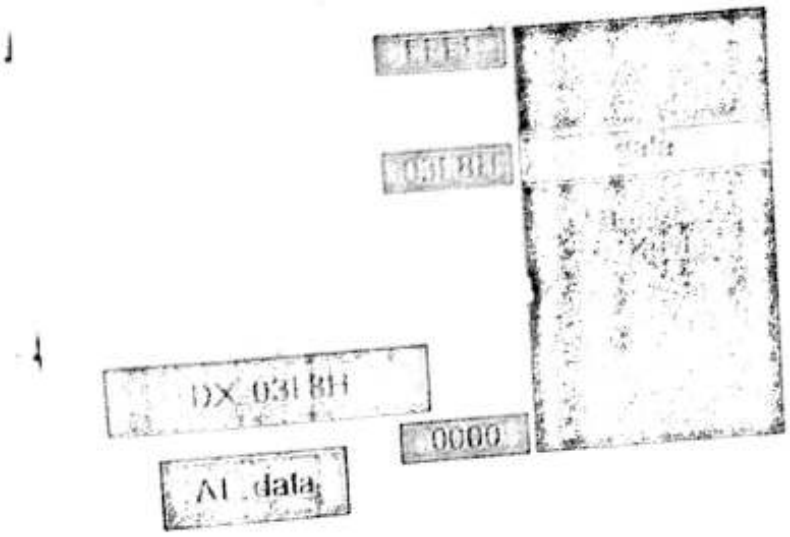
```
IN   AL,2EH      ;AL  ←—  port 2EH
IN   AX,26H      ;AL  ←—  port 26H
                 ;AH  ←—  port 27H

IN   AL,DX       ;AL  ←—  port DX
IN   AX,DX       ;AL  ←—  port DX
                 ;AH  ←—  port DX+1

OUT  2EH,AL      ; port 2EH  ←—  AL
OUT  2EH,AX      ; port 2EH  ←—  AL
                 ; port 2FH  ←—  AH

OUT  DX,AL       ; port DX   ←—  AL
OUT  DX,AX       ; port DX   ←—  AL
                 ; port DX+1 ←—  AH
```

Example

The serial port is at 03F8H(>255), read a byte from the serial port.

Since 03F8H>255

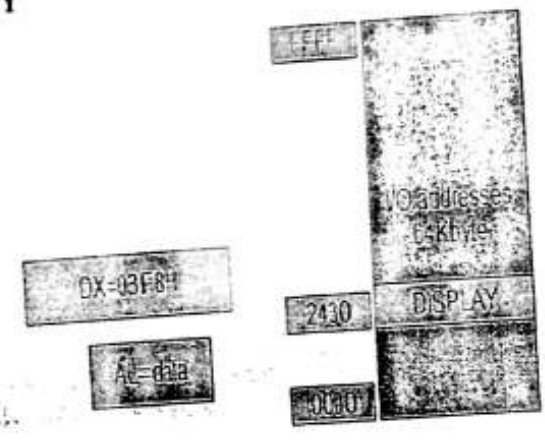
```
MOV DX,03F8H
IN AL,DX
```



Example

Define the DISPLAY device at 2430H and read a byte from it.

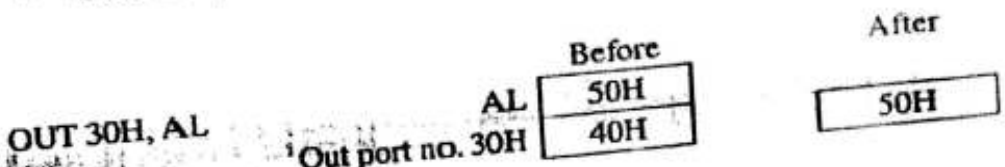
```
DISPLAY EQU 2430H
MOV DX,DISPLAY
IN AL,DX
```

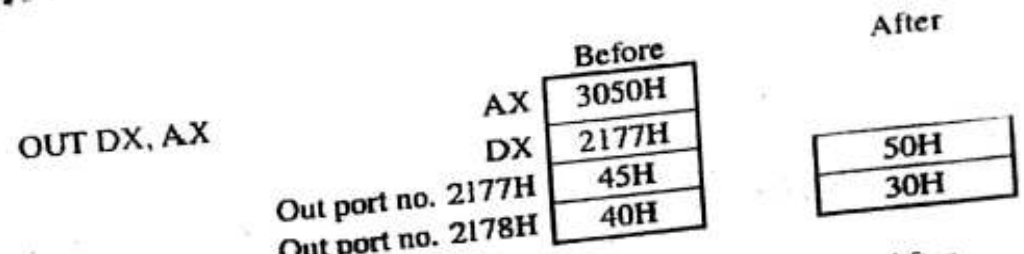


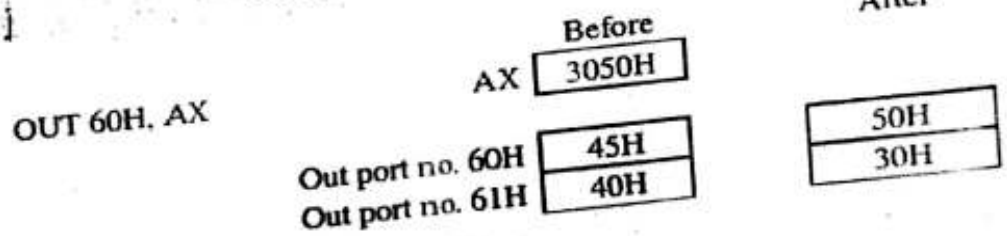
OUT Instruction

Syntax:

- OUT port address, AL ; can be AX or AL
- OUT DX,AL ; Just holds for DX







Example:

Send char 'A' to the printer at 03A0H.

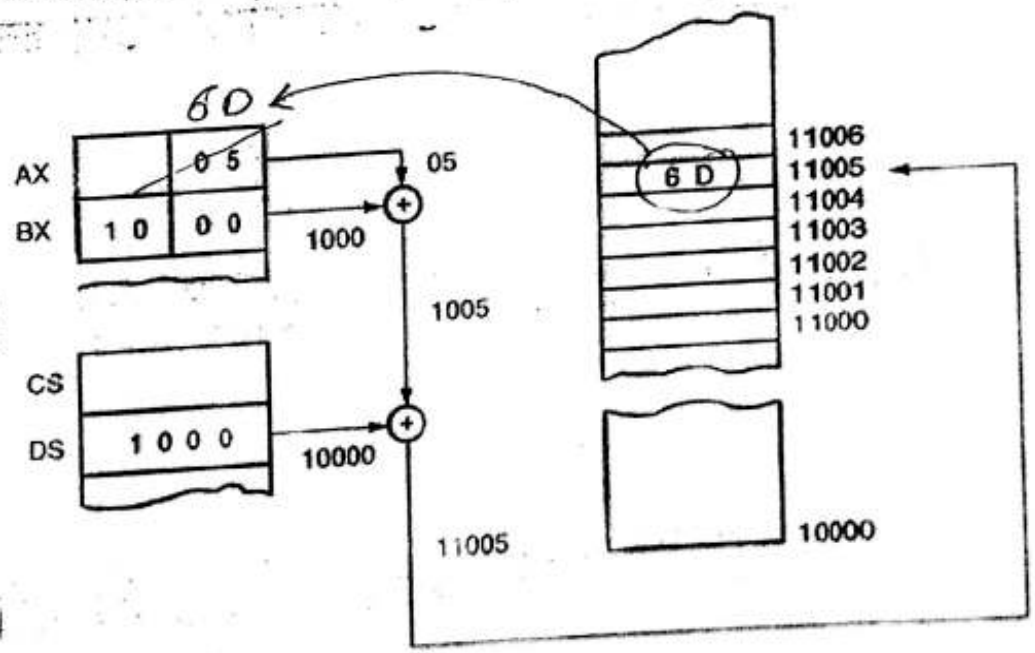
```
MOV DX,03A0H
MOV AL,'A'
OUT DX,AL
```

XLAT Instruction

XLAT: Translate byte using look up table

Mnemonic	Meaning	Format	Operation	Flags affected
XLAT	Translate	XLAT	((AL)+(BX)+(DS)0)→(AL)	None

- Replace the data in AL with a data in a user defined **look-up table (!!!)**
- BX stores the beginning address of the table.
- At the beginning of the execution, the number in AL is used as the index of the look-up table.
- It does not modify flags
- XLAT is the only instruction that adds 8 bit number to 16 bit number.
- XLAT called translate instruction, the operation of XLAT is changing the value of AL with the memory pointed to by (AL+BX).



Example:

Assume (DS) = 0300H, (BX)=0100H, and (AL)=0DH

XLAT replaces contents of AL by contents of memory location with

PA=(DS)0 +(BX) +(AL)

= 03000H + 0100H + 0DH = 0310DH

Thus

(0310DH) → (AL)

- The translate (XLAT) data transfer instruction is shown below. It can be used for say an ASCII to EBCDIC code conversion.

- The content of BX represents the offset of the starting address of the look up table from the beginning of the current data segment while the content of AL represents the offset of the element which is to be accessed from the beginning of the look up table.

As an example, let DS = 0300 H, BX = 1234 H and AL = 05 H.

Hence, PA = 03000 H + 1234 H + 05 H = 04239 H

Thus, execution of XLAT would put the content of 04239 H into AL register. Conceptually, the content of 04239 H in EBCDIC should be the same as the ASCII character equivalent of 05 H.

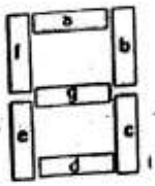
EX

Suppose that a seven segment LED display lookup table is stored in memory at address TABLE, now XLAT can use this table to translate the BCD number in AL to a seven segment code also in AL.

Let TABLE =1000H, DS=1000H,

a	b	c	d	e	f	g	
1	1	1	1	1	1	0	
0	1	1	0	0	0	0	
1	1	0	1	1	0	1	
1	1	1	1	0	0	1	
0	1	1	1	0	1	1	
1	0	1	1	0	1	1	
1	0	1	1	1	1	1	
1	1	1	0	0	0	0	
1	1	1	1	1	1	1	
1	1	1	1	0	1	1	

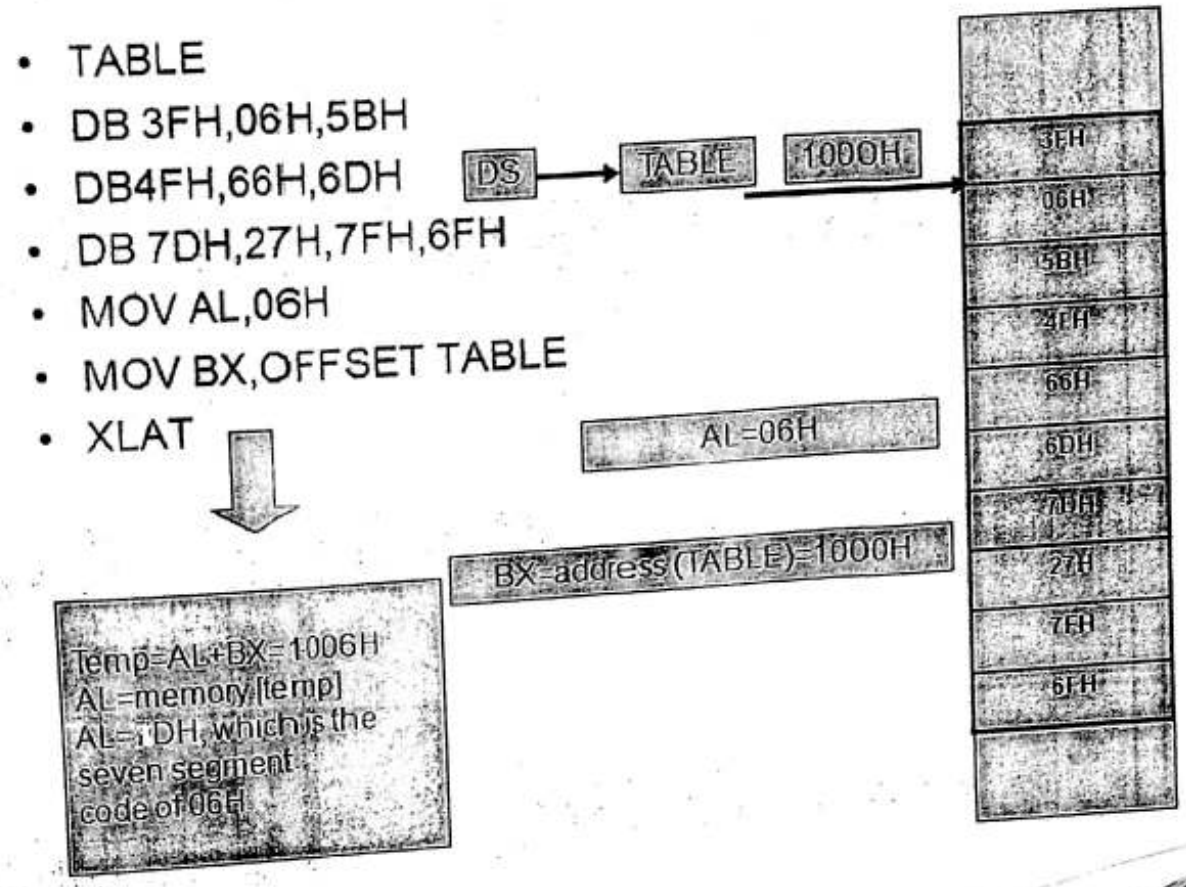
0	3FH
1	06H
2	5BH
3	4FH
4	66H
5	6DH
6	7DH
7	27H
8	7FH
9	6FH



The code for the operation of converting is very simple using XLAT

- TABLE DB 3FH,06H,5BH,4FH,66H,6DH,7DH,27H,7FH,6FH
- MOV AL,SOME_BCD_VALUE
- MOV BX,OFFSET TABLE
- XLAT
- For example take 06H(6 BCD) for the initial value of AL, after translation AL contains 7DH,

- TABLE
- DB 3FH,06H,5BH
- DB4FH,66H,6DH
- DB 7DH,27H,7FH,6FH
- MOV AL,06H
- MOV BX,OFFSET TABLE
- XLAT



EX

Write code to read the input key from port (60H) and output the value to a 7-segemnt display at 80H.

```
KEYS EQU 60H;
DISPLAY EQU 80H;
TABLE DB 3FH, 06H,5BH,4FH,66H,6DH,7DH,27H,7FH,6FH
LEA BX,TABLE;
IN AL,KEYS;
XLAT
OUT DISPLAY,AL
```

PUSH and POP Instructions

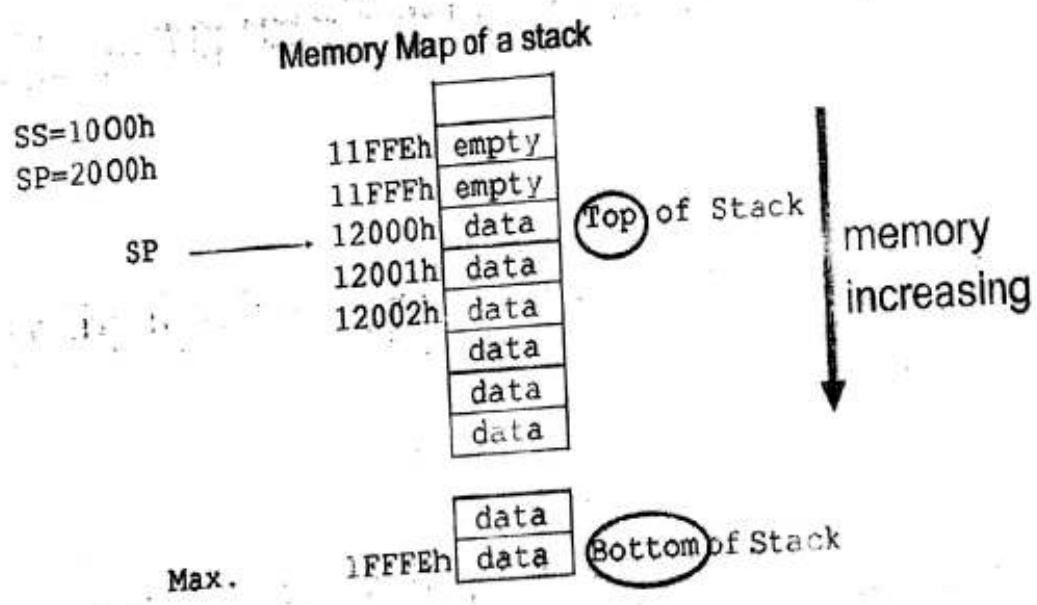
The stack is used as temporary storage of registers and memory locations
PUSH is used to store data and POP is used to retrieve it back from the stack

Mnemonic	Meaning	Format	Operation	Flags Affected
PUSH	Push word onto stack	PUSH S	((SP)) ← (S) (SP) ← (SP)-2	None
POP	Pop word off stack	POP D	(D) ← ((SP)) (SP) ← (SP)+2	None

Operand (S or D)
Register
Seg-reg (CS illegal)
Memory

- The stack is a block of memory reserved for temporary storage of data and registers. Access is LAST-IN, FIRST-OUT (LIFO)
- The last memory location used in the stack is given by (the effective address calculated from the SP register) and (the SS register):

Example:



PUSH: Push to stack

- It pushes the operand into top of stack.
- This instruction pushes the contents of the specified register/memory location on to the stack.
- The stack pointer is decremented by 2, after each execution of this instruction.

Example

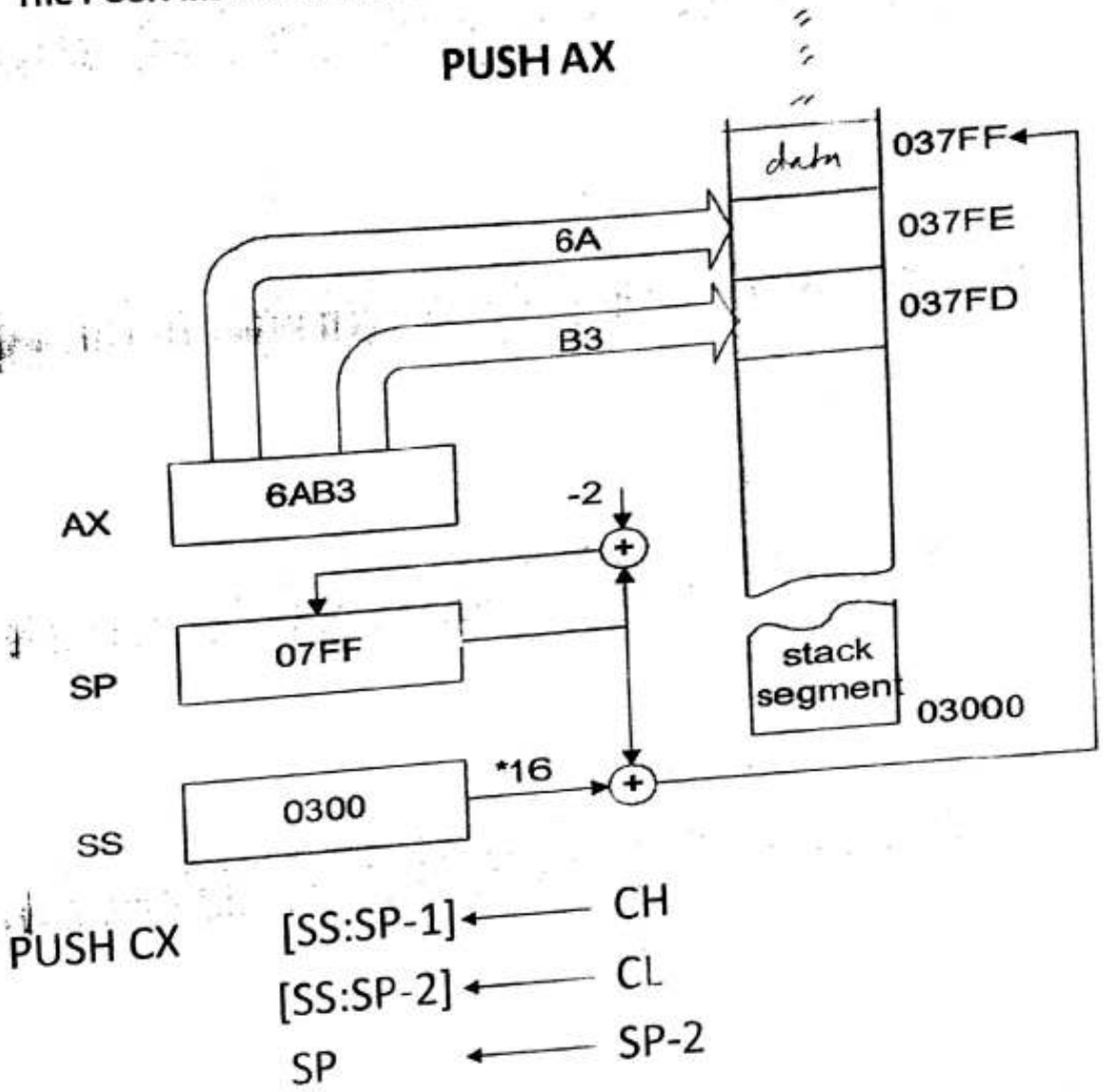
PUSH AX	Decrement SP by 2 and copy AX to stack
PUSH DS	Decrement SP by 2 and copy DS to stack
PUSH [5000H]	Decrement SP by 2 and copy word from memory in DS

POP : Pop from stack

- It pops the operand from top of stack to Destination.
- The stack pointer is incremented by 2.
- Destination can be a general purpose register, segment register (except CS) or memory location.
- This instruction when executed, loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer.

```
POP AX
POP DS
POP [5000H]
```

The PUSH instruction transfers two bytes to the top of the stack.



PUSHF (No operand)

The **PUSHF** (push flag) instruction copies the contents of the flag register to the stack.

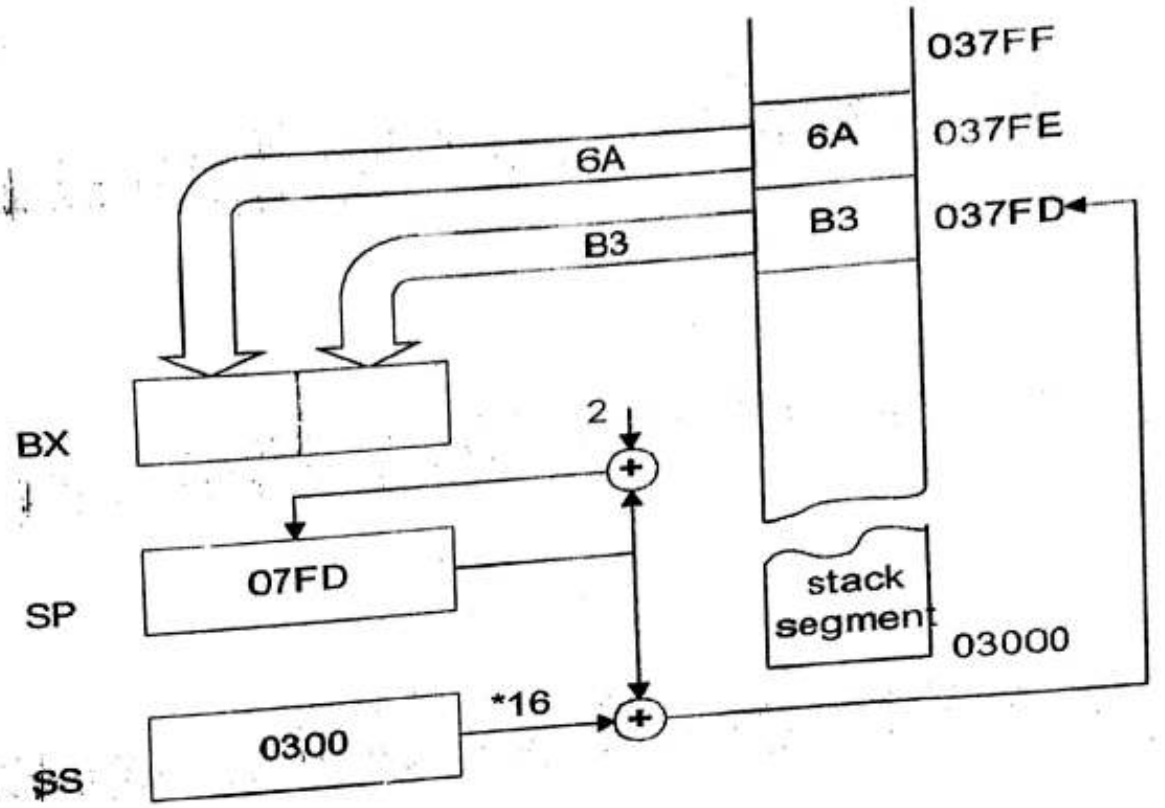
Algorithm:

SP = SP - 2
SS:[SP] (top of the stack) = flags

C	Z	S	O	P	A
unchanged					

The POP instruction performs the inverse operation of a PUSH instruction.

POP BX



POP BX

BL ← [SS:SP]
BH ← [SS:SP+1]
SP ← SP+2

POPF (No operand) **Pops the stack top to flag register.**

Algorithm:

flags = SS:[SP] (top of the stack)
SP = SP + 2

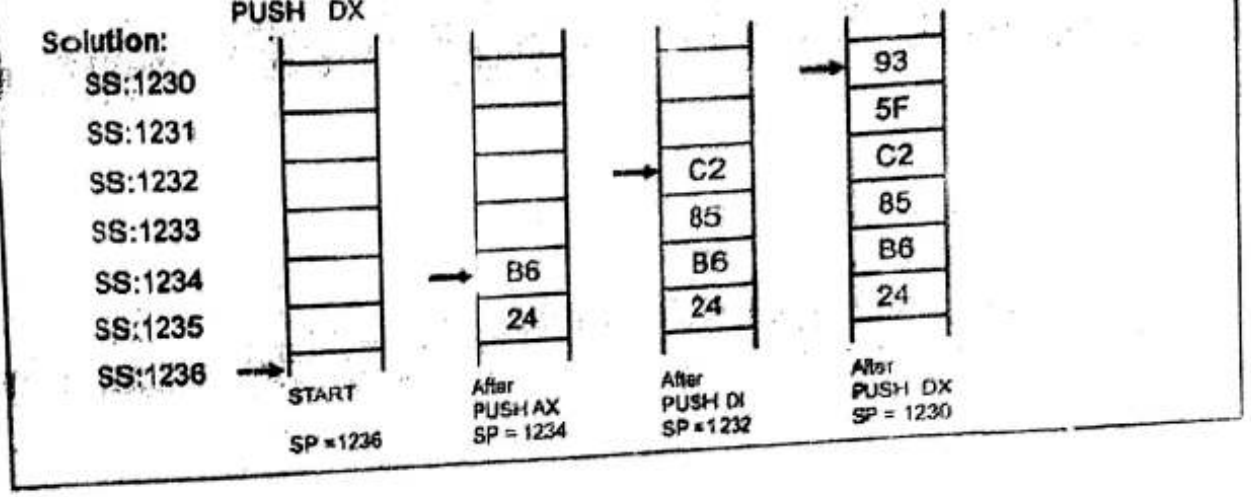
C	Z	S	O	P	A
popped					

Ex:

Assuming that SP = 1236, AX = 24B6, DI = 85C2, and DX = 5F93, show the contents of the stack as each of the following instructions is executed:

```
PUSH AX
PUSH DI
PUSH DX
```

Solution:



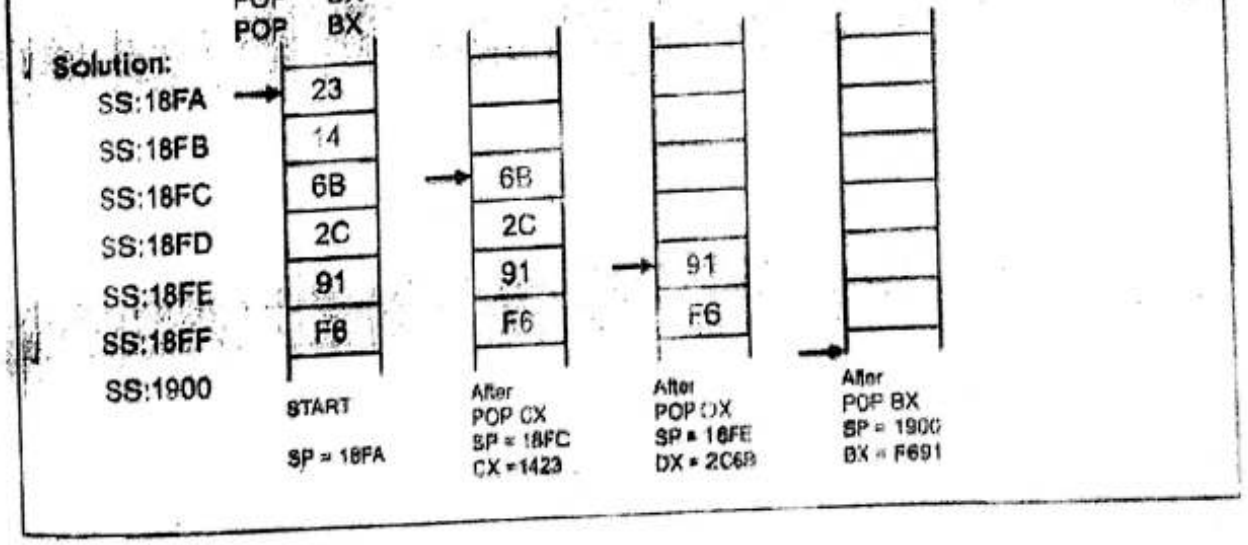
Popping to stack

Ex:

Assuming that the stack is as shown below, and SP = 18FA, show the contents of the stack and registers as each of the following instructions is executed:

```
POP CX
POP DX
POP BX
```

Solution:



Ex Given SS=0105h , What is the outcome of the instruction

PUSH AX

BOS = 01050+FFFEh=1 104h

SP=0008h

AX=1234h

TOS=01050+0008h=1058H

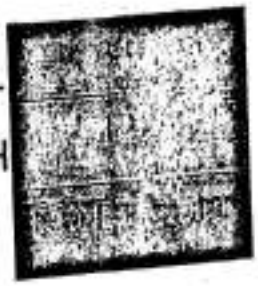
Decrement the SP by 2 and write AX into the word location 1056h

SP

SS:0006 1056h AL

SS:0007 1057h AH

SS:0008 1058h



EX What is the outcome of the following instructions

POP AX

POP BX

If originally (SS:SP)=1056H

1056	34
1057	12
1058	BB
1059	AA

Read into the specified register from the stack and increment the stack pointer for each POP operation

At the first POP (AX) = 1234h (SS:SP) = 1058h

At the second POP (BX) = AABBh and (SS:SP) =105AH